2. การค้นหาข้อมูล (Search)

การค้นหาข้อมูลที่สำคัญมี 2 รูปแบบคือ (1) การค้นหาเชิงเส้น (linear search) และ (2) การค้นหาแบบทวิภาค (binary search) ซึ่งทั้ง 2 รูปแบบก็มีฟังก์ชันจาก STL ให้เรียกใช้ หรือจะเขียนโปรแกรมของขั้นตอนวิธีค้นหาเองก็ทำได้

- Linear search สามารถค้นหาได้ โดยไม่มีเงื่อนไขเกี่ยวกับข้อมูล และมี time complexity $O(N)$
- Binary search สามารถค้นหาได้ เมื่อข้อมูลมี<u>การเรียงลำดับแล้ว</u>เท่านั้น และมี time complexity $O(\log N)$

การค้นหาเชิงเส้น (linear search) → ไม่ควรเขียนเอง

ตาราง 2.1 ฟังก์ชันเกี่ยวกับการค้นหาเชิงเส้นจาก STL

Function	Description	Time Complexity
`all_of(l,r,f)` ∀	Test condition defined by `f` on all elements in range `[l,r)`	$\theta(N)$
`any_of(l,r,f)` ∈	Test if any element in range Test if any element in range `[l,r)` fulfills condition defined by `f` on all elements	$\theta(N)$
เช่น ธรรมดา ← `count(l,r)`	Count appearances of value in range `[l,r)`	$\theta(N)$
`count_if (l,r,f)` → เงื่อนไข	Return number of elements in range `[l,r)` satisfying condition defined by `f`	$\theta(N)$
`find(l,r)`	Find value in range `[l,r)`	$\theta(N)$
`find_if l,r,f)`	Find element in range `[l,r)` satisfying condition defined by `f`	$\theta(N)$
`max_element(l,r)`	Return largest element in range `[l,r)`	$\theta(N)$
`min_element(l,r)`	Return smallest element in range `[l,r)`	$\theta(N)$

โปรแกรมที่ 2.1 การใช้ฟังก์ชัน find( ), find_if( ), count( ) และ count_if( ) กับข้อมูลในอาร์เรย์ และ vector

```
1.   #include<iostream>
2.   #include<algorithm>
3.   #include<vector>     // vector
4.   using namespace std;
5.
6.   int main(){
7.       // find(),find_if(),count(),count_if
8.       // works with array, STL containers as follows
9.       // vector,array,deque,list,set,unordered_set
10.      int x = 3, N = 4;
11.      int arr[] = {2,4,2,4};
12.      vector<int> vec = {1,3,1,3};
13.      auto is_even = [](int i){ return i%2 == 0; };
14.
15.      cout << "Array: {2,4,2,4}\n";
16.      if(find(arr,arr+N, x)!=arr+N)
17.          cout << "   " << x << " found\n";
18.      else
19.          cout << "   " << x << " not found\n";
20.      if(find_if(arr,arr+N,is_even)!=arr+N)
21.          cout << "   Even found in array\n";
22.      else
23.          cout << "   Even not found in array\n";
24.      cout << "   Count of " << x << " => "
25.           << count(arr,arr+N,x) << "\n";
26.      cout << "   Count of even => "
27.           << count_if(arr,arr+N,is_even) << "\n";
28.
29.      cout << "Vector: {1,3,1,3}\n";
30.      if(find(vec.begin(),vec.end(), x)!=vec.end())
31.          cout << "   " << x << " found\n";
32.      else
33.          cout << "   " << x << " not found\n";
34.
35.      if(find_if(vec.begin(),vec.end(),is_even)!=vec.end())
36.          cout << "   Even found in vector\n";
37.      else
38.          cout << "   Even not found in vector\n";
39.      cout << "   Count of " << x << " => "
40.           << count(vec.begin(),vec.end(),x) << "\n";
41.      cout << "   Count of even => "
42.           << count_if(vec.begin(),vec.end(),is_even) << "\n";
43.      return 0;
44.  }
45.
46.
```

ผลลัพธ์

```
Array: {2,4,2,4}
   3 not found
   Even found in array
   Count of 3 => 0
   Count of even => 4
Vector: {1,3,1,3}
   3 found
   Even not found in vector
   Count of 3 => 2
   Count of even => 0
```

โปรแกรม 2.1 แสดงการทำงานของฟังก์ชัน find( ) และ find_if( ) เพื่อค้นหาข้อมูล และฟังก์ชัน count() และ count_if( ) เพื่อนับจำนวนข้อมูล นอกจากอาร์เรย์และ vector แล้ว ฟังก์ชันทั้ง 4 สามารถค้นหาและนับข้อมูลที่เก็บใน (STL) `array, deque, list, set` และ `unordered_set`

ฟังก์ชัน find_if( ) และ count() ต้องรับค่าพารามิเตอร์ตัวที่ 3 ซึ่งเป็นฟังก์ชันที่ใช้ในการกำหนดเงื่อนไขของข้อมูลที่จะค้นหาหรือนับ ฟังก์ชันนี้สามารถเขียนเป็นฟังก์ชันแบบปกติ หรือเขียนเป็น lambda function ดังที่แสดงในบรรทัดที่ 14 ก็ได้

โปรแกรมที่ 2.2 การใช้ฟังก์ชัน min_element( ), max_element( ), all_of( ) และ any_of( ) กับข้อมูลในอาร์เรย์ และ vector

```
1.   #include<iostream>
2.   #include<algorithm>
3.   #include<vector>     // vector
4.   using namespace std;
5.
6.   int main(){
7.       // min_elment(),max_element(),all_of(),any_of
8.       // works with array, STL containers as follows
9.       // vector,array,deque,list,set,unordered_set
10.
11.
```

```
12.      int x = 3, N = 4;
13.      int arr[] = {2,4,2,4};
14.      vector<int> vec = {1,3,1,3};
15.      auto lessthan_4 = [](int i){ return i < 4; };
16.
17.      cout << "Array: {2,4,2,4}\n";
18.      cout <<"   Min => "<<*min_element(arr,arr+N)<<" \n";
19.      if(any_of(arr,arr+N, lessthan_4))
20.          cout << "   Exist some element less than 4\n";
21.      else
22.          cout << "   No elements less than 4\n";
23.
24.      cout << "Vector: {1,3,1,3}\n";
25.      cout <<"   Max => "
26.           <<*max_element(vec.begin(),vec.end())<<" \n";
27.      if(all_of(vec.begin(),vec.end(), lessthan_4))
28.          cout << "   All elements less than 4\n";
29.      else
30.          cout << "   Not all elements less than 4\n";
31.      return 0;
32.  }
```

ผลลัพธ์

```
Array: {2,4,2,4}
   Min => 2
   Exist some element less than 4
Vector: {1,3,1,3}
   Max => 3
   All elements less than 4
```

โปรแกรม 2.2 แสดงการทำงานของฟังก์ชัน min_element( ) และ max_element( ) เพื่อค้นหาข้อมูลที่มีค่าน้อยที่สุด และค่ามากที่สุดตามลำดับ ส่วนฟังก์ชัน all_of( ) จะตรวจสอบว่าข้อมูลทุกตัวในอาร์เรย์ หรือ STL container มีสมบัติตามที่ระบุด้วยฟังก์ชันที่รับเป็นพารามิเตอร์ตัวที่ 3 หรือไม่ และฟังก์ชัน any_of( ) จะตรวจสอบว่ามีข้อมูลอย่างน้อย 1 ตัวที่มีสมบัติตามที่ระบุด้วยฟังก์ชันที่รับเป็นพารามิเตอร์ตัวที่ 3 หรือไม่

ฟังก์ชันทั้ง 4 สามารถทำงานกับข้อมูลที่เก็บในอาร์เรย์ และ STL container ต่อไปนี้ `array`, `vector`, `deque`, `list`, `set` และ `unordered_set`

การค้นหาแบบทวิภาค (binary search)

การค้นหาแบบทวิภาคหรือ binary search เป็นการค้นหาที่มี time complexity น้อยกว่าการค้นหาเชิงเส้น (linear search) อย่างมีนัยยะสำคัญ อย่างไรการค้นหาด้วย binary search นั้นมีเงื่อนไขว่าข้อมูลต้องเรียงลำดับมากก่อน ซึ่งโดยปริยายจะเรียงจากน้อยไปมาก (ascending) binary search จะหาข้อมูลที่อยู่ตำแหน่ง (index) กลางระหว่างข้อมูลที่ตำแหน่งแรกและข้อมูลที่ตำแหน่งสุดท้าย จากนั้น

1. ถ้าข้อมูลที่ค้นหามีค่าเท่ากับข้อมูลที่ตำแหน่งกลาง binary search binary search จะคืนตำแหน่งกลาง
2. ถ้าข้อมูลที่ค้นหามีค่าน้อยกว่าข้อมูลที่ตำแหน่งกลาง binary search จะค้นหาข้อมูลตั้งแต่ตำแหน่งแรกจนถึงข้อมูลที่อยู่ก่อนตำแหน่งกลาง 1 ตำแหน่ง
3. ถ้าข้อมูลที่ค้นหามีค่ามากกว่าข้อมูลที่ตำแหน่งกลาง binary search จะค้นหาข้อมูลที่อยู่หลังตำแหน่งกลาง 1 ตำแหน่ง จนถึงข้อมูลที่อยู่ตำแหน่งสุดท้าย

การเขียนโปรแกรมการค้นหาด้วย binary search สามารถเขียนแบบ iterative หรือ recursive ก็ได้ อย่างไรก็ตาม การเขียนแบบ iterative จะช่วยลดความเสี่ยงของการเกิด stack overflow ได้ time complexity ของ binary search ได้มาจาก recurrence relation $T(N) = T\left(\frac{N}{2}\right) + \theta(1)$ ทำให้ได้ big-O เป็น $O(\log N)$

รหัสเทียมของขั้นตอนวิธี binary search แบบ iterative

```
procedure binarySearch(x:integer,a1, a2, …, an:increasing integers)
i := 1   {i is left endpoint of search interval}
j := n   {j is right endpoint of search interval}
while (i < j)
      m := floor((i + j)/2)
      if x > am then i := m + 1
      else j := m
if x = ai then location := i
else location := -1
return location
```

ใช้ binary search ค้นหา **x = 16** (**หยุดค้นหาเมื่อ i >= j**)

รอบที่ 1 **i = 1** **j = 10** **m = floor((1+10)/2) = 5**

ลำดับ	5	7	10	11	12	14	16	17	20	21
ตำแหน่ง	1	2	3	4	5	6	7	8	9	10

พิจารณาค่าที่ตำแหน่ง 5 คือ 12 พบว่า 16 > 12 ปรับค่า **i = m + 1 = 6**

รอบที่ 2 **i = 6** **j = 10** **m = floor((6+10)/2) = 8**

ลำดับ	5	7	10	11	12	14	16	17	20	21
ตำแหน่ง	1	2	3	4	5	6	7	8	9	10

พิจารณาค่าที่ตำแหน่ง 8 คือ 17 พบว่า **16** <= 17 ปรับค่า **j = m = 8**

รอบที่ 3 **i = 6** **j = 8** **m = floor((6+8)/2) = 7**

ลำดับ	5	7	10	11	12	14	16	17	20	21
ตำแหน่ง	1	2	3	4	5	6	7	8	9	10

พิจารณาค่าที่ตำแหน่ง 7 คือ 16 พบว่า 16 <= 16 ปรับค่า **j = m = 7**

รอบที่ 4 **i = 6** **j = 7** **m = floor((6+7)/2) = 6**

ลำดับ	5	7	10	11	12	14	16	17	20	21
ตำแหน่ง	1	2	3	4	5	6 6	7	8	9	10

พิจารณาค่าที่ตำแหน่ง 6 คือ 14 พบว่า 16 > 14 ปรับค่า **i = m + 1 = 7**

รอบที่ 5 ไม่มี เพราะ **i >= j (7 >= 7)**
จากนั้น ตรวจสอบว่า ค่าที่ตำแหน่ง **i = 7** เท่ากับ **16** หรือไม่
ถ้าใช่ `return location = i = 7`
ถ้าไม่ใช่ `return -1`

รูปที่ 1

รูปที่ 1 แสดงตัวอย่างการค้นหาด้วย binary search แบบ iterative ถ้าค้นพบ จะคืนตำแหน่งของข้อมูล แต่ถ้าค้นไม่พบจะคืนค่า -1

รหัสเทียมของขั้นตอนวิธี binary search แบบ recursive

```
procedure binarySearch(i,j,x :integers,1≤i ≤j ≤n
                        a1, a2, …, an:increasing integers)
m := floor((i + j)/2)
if x = am then
     return m
else if x < am and i < m
     return binarySearch(i, m-1, x)
else if x > am and j > m
     return binarySearch(m+1, j, x)
else
     return -1
```

ตาราง 2.2 ฟังก์ชันเกี่ยวกับการค้นหาแบบทวิภาคจาก STL

Function	Description	Time Complexity
`binary_search (l,r,x)`	Test if value `x` exists in sorted sequence in range `[l,r)`	$\theta(\log N)$ random access iterator $\theta(N)$ non -random access iterator
`lower_bound (l,r,x)`	`Return iterator to the first element` in range `[l,r) which >= x`	เหมือน `binary_search( )`

ฟังก์ชัน binary_search( ) จะคืนเพียงค่าความจริงว่าค้นพบหรือไม่ แต่หากต้องการที่อยู่ในหน่วยความจำ (memory address) ของค่าที่ค้นจะต้องใช้ ฟังก์ชัน lower_bound( ) ในกรณีที่ต้องการตำแหน่ง (index) ของค่าที่ค้นในอาร์เรย์หรือ vector ที่บรรจุข้อมูล จะต้องนำ memory address ของค่าที่ค้นลบด้วย memory address ของค่าแรกที่อยู่ในอาร์เรย์หรือ vector ที่บรรจุข้อมูล หากผลลบนี้มากกว่าหรือเท่ากับจำนวนข้อมูลในอาร์เรย์หรือ vector แสดงว่าค้นไม่พบ

ในกรณีที่อาร์เรย์หรือ vector บรรจุค่าที่ค้นมากกว่า 1 ตัว ฟังก์ชัน lower_bound( ) จะคืน memory address ของตัวแรกที่ค้นพบ

โปรแกรมที่ 2.3การใช้ฟังก์ชัน binary_search( ) และ lower_bound( )

```
1.   #include<iostream>
2.   #include<algorithm>
3.   #include<vector>    // vector
4.   using namespace std;
5.   int main(){
6.       int N = 4;
7.       int arr[] = {1,2,2,5};
8.       vector<int> vec = {1,2,2,5};
9.       vector<int> x = {2,5,8};  // elem to find
10.
11.      cout << "Array: {1,2,2,5}\n";
12.      for(auto &i : x){
13.          if(binary_search(arr,arr+N, i))
14.              cout << "  Found " << i << "\n";
15.          else
16.              cout << "  Not found " << i << "\n";
17.      }
18.
19.      cout << "Vector: {1,2,2,5}\n";
20.      for(auto &i : x){
21.          auto pos=lower_bound(vec.begin(),vec.end(),i)
22.                   - vec.begin();
23.          if(pos < vec.size())
24.              cout <<"  Found " << i << " at index " << pos << "\n";
25.          else
26.              cout << "  Not found " << i << "\n";
27.      }
28.      return 0;
29.  }
30.
```

ผลลัพธ์

```
Array: {1,2,2,5}
  Found 2
  Found 5
  Not found 8
Vector: {1,2,2,5}
  Found 2 at index 1
  Found 5 at index 3
  Not found 8
```

โปรแกรมที่ 1.3A การใช้ฟังก์ชัน sort( ) ในการเรียงข้อมูลชนิด Student โดยเรียงด้วยคะแนนสอบครั้งที่ 2 แต่ถ้าคะแนนสอบครั้งที่ 2 เท่ากันใช้คะแนนสอบครั้งที่ 3 (เรียงจากมากไปน้อย)

```
1.   #include<iostream>
2.   #include<algorithm>   // sort
3.   #include<string>      // string
4.   using namespace std;
5.   // Forwards
6.   template <typename T>
7.   void printStudentArray(const T arr[], const int &n);
8.   class Student{
9.       public:
10.      int id;
11.      string name;
12.      int scores[3];
13.      // constructor
14.      Student(int id, string name, int s0, int s1, int s2){
15.          this->id = id; this->name = name;
16.          this->scores[0] = s0; this->scores[1] = s1;
17.          this->scores[2] = s2;
18.      }
19.      int getTotalScore(){
20.          return scores[0]+scores[1]+scores[2];
21.      }
22.      // overload operator< to sort by scores[1]
23.      // then scores[2] (descending)
24.      bool operator<(Student b){
25.          if(this->scores[1] == b.scores[1])
26.              return this->scores[2] > b.scores[2];
27.          return this->scores[1] > b.scores[1];
28.      }
29.      void printStudent() const{
30.          cout<<"("<<id<<","<<name<<"=>["<<
31.                scores[0]<<","<<scores[1]<<","<<scores[2]<<"])";
32.      }
33.  };
34.  int main(){
35.      int N = 5;
36.
37.      Student arr[] = { Student(1,"Emily", 10,9,10),
38.                        Student(2,"Daisy",10,10,10),
39.                        Student(3,"April",8,8,8),
40.                        Student(4,"Lizzy", 7,8,9),
41.                        Student(5,"Jenna", 9,8,7) };
42.
43.
44.
45.
```

```
    cout << "Before sort\n";
    printStudentArray(arr,N);
    sort(arr,arr+N);
    cout << "After sort by 2nd score, then by 3rd score
            (descending)\n";
    printStudentArray(arr,N);
    return 0;
}
// print array of class Student
template <typename T>
void printStudentArray(const T arr[], const int &n){
    for(int i=0; i<n; i++){
        arr[i].printStudent();
        cout << "\n";
    }
}
```

ผลลัพธ์

```
Before sort
(1,Emily=>[10,9,10])
(2,Daisy=>[10,10,10])
(3,April=>[8,8,8])
(4,Lizzy=>[7,8,9])
(5,Jenna=>[9,8,7])
After sort by 2nd score, then by 3rd score (descending)
(2,Daisy=>[10,10,10])
(1,Emily=>[10,9,10])
(4,Lizzy=>[7,8,9])
(3,April=>[8,8,8])
(5,Jenna=>[9,8,7])
```